Technical Report on Navigation – Using Sensor Fusion Approach

Özkil, Ali Gürcan

Publication date:
2009

Technical Report on
Navigation – Using Sensor Fusion Approach

Ali Gürcan Özkil

September 2009

Table of Contents

1 Objective

This report aims to describe the navigation system of the Nestor project. It summarizes the existing navigation methods and outlines the improved approach which relies on sensor fusion.

2 Introduction

A robust, scalable and adaptable navigation system has always been the focus of attention in mobile robot applications. The project Nestor, being no exception, aims to improve on these properties while avoiding over-complexity; which tends to hinder the wide application of mobile robots.

In line with this purpose, a new navigation approach is proposed. This approach utilizes the strengths of two most commonly used exteroceptive sensors ([1]) in robotics: Laser range finders and cameras. The system also combines navigation with metric maps based on laser range finder readings and navigation with topological maps based on “places of interests” that are extracted from captured images.

Rest of the report is structured as follows. In the next section, existing navigation system is explained. The following section introduces the improvement made on the existing system. Finally, concluding remarks are presented

3 Robot Navigation - Using Metric Maps and Laser Range Finders

Before going into the details, it is better to sketch the practical implementation and the software structure of the existing software architecture, since it is the best way to describe the metric navigation system of Nestor.

A robot is a complex mechatronic system which is composed of many hardware and software components that needs to be orchestrated in (near) real time. Therefore, a software architecture is needed to synchronize simultaneously running processes and abstract hardware components. Luckily, there exist a few free or open source projects that fully or partially address these issues.

3.1 The Software Architecture

Among other alternatives such as Microsoft Robotics Studio (MSRS) [2], Player/Stage [3] or Mobile Robot Programming Toolkit (MRPT) [4]; Carnegie Mellon Navigation Toolkit (CARMEN) [5] had been selected as the platform to improve upon at the beginning of the project. The main reason for selection was because CARMEN mainly

focuses on the navigation problem instead of providing a broad and generic robotics software platform, and it natively supported the experimental hardware that was selected for the project (Pioneer 3-Dx robots from ActivMedia[6]).

3.1.1 CARMEN

CARMEN software platform is targeted for Linux operating systems and it was developed using C programming language. It is heavily based on a client-server architecture, where the navigation task is decomposed into several smaller tasks and handled by individual clients. There are a few advantages of this approach, but mainly, it facilitates a distributed system and it permits failsafe operation where only the related processes are affected in case of the failure of a component.

A central process -the server is responsible from inter-process communication (IPC), where clients publish and request information through. Under the hood, IPC takes care of opening and closing TCP/IP sockets, registering, sending and receiving packets, serializing/de-serializing the data; and therefore provides a high-level support for creating client/server systems. Figure 1 shows the operating principle of IPC.

Modules of the CARMEN can be grouped in four categories: Essential processes, primary processes, navigational processes and complementary processes. Figure 2 and Figure 3 shows the relationships between these processes.

3.1.1.1 Essential Processes

The main process in the CARMEN toolkit is the "*central*", which basically is an IPC process where all sorts of information that are produced by sensors, actuators and software components pass through.

Along with the *central*, the *parameter_server* is the second process that needs to be running all the time for the other processes to function. This process is responsible from serving a number of parameters that are needed by several other processes. These parameters are numerous, and they vary from hardware settings such as the communication ports to be opened and the speed of communication, to operational settings such as maximum permissible speed of the robot or the number of particles to be used in the localization process. All of the parameters are stored in a single text file and this file is passed as an argument to the *parameter_server* process at runtime. Contents of a typical parameter file can be seen in Appendix I.

3.1.1.2 Primary Processes

On top of the essential processes there are a number of primary processes that needs to be run for the actual functionality of the system.

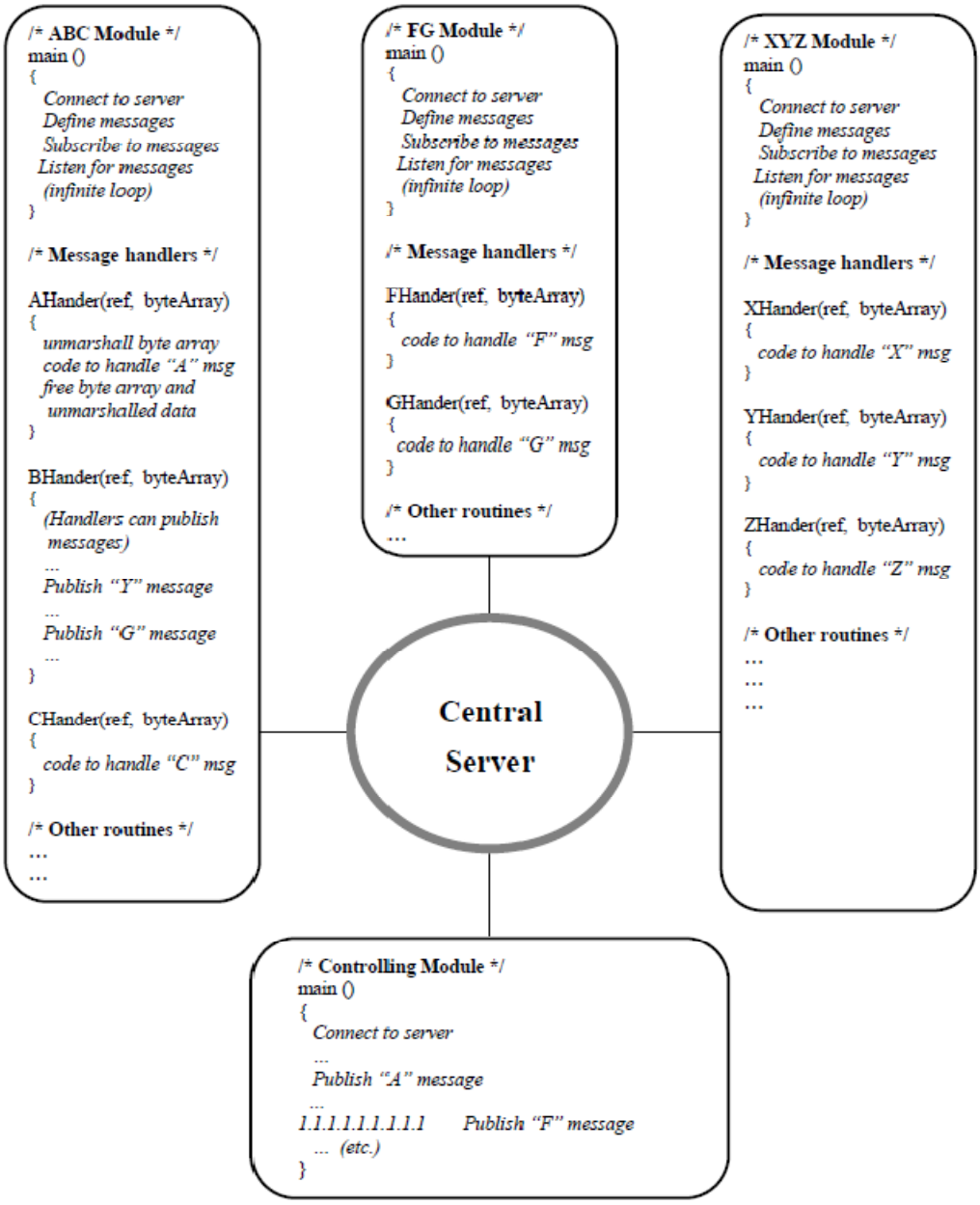

Figure 1, Inter Process Communication Architecture

The *robot* process is responsible from providing the core functionality. It abstracts the hardware of the robot by providing an interface to a generic robot object. It also provides a collision avoidance system which stops the robot in front of the obstacles.

The *pioneer* process belongs to a family of processes that handles the actual communication between actual robot base and the *robot* process. It translates the

high level motion commands that are given to the robot to low level motor actuation commands, and serves proprioceptive information (such as odometry readings, battery state etc.) to the *robot* process. There are two other similar processes: *iRobot-Atr* and *nomadic* which are used for the same purpose as the *pioneer,* but are used to interface with other robot hardware base platforms.

The *laser* process interfaces a real laser sensor to the framework and publishes range readings that come from laser(s). A mobile robot can utilize multiple lasers which can be of different types (Either SICK LMS or Hokuyo URG series). Positions, orientations and other properties of the laser(s) are specified in the parameter.ini file.

These three processes are the primary processes that make it possible to command a physical robot through CARMEN platform. At this point, the *robot* process can be used to pass motion commands to the robot by direct user input through i.e. computer keyboard.

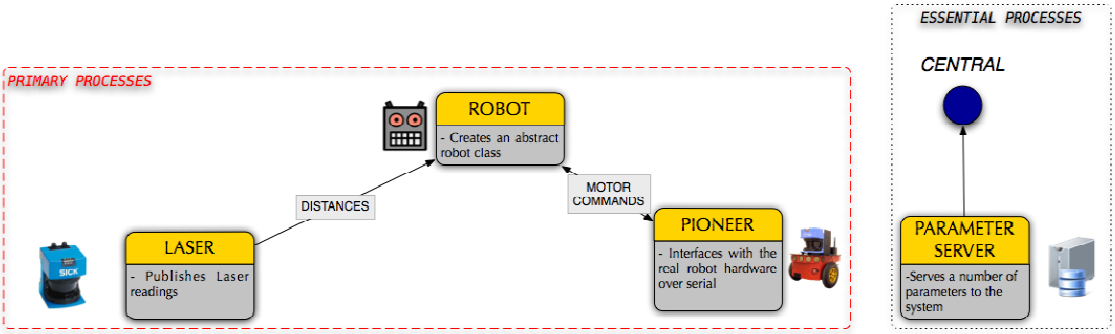

Figure 2, Essential and Primary Processes

3.1.1.3 Navigational Processes

The navigation problem can be solved by answering three repeatedly stated problems: *(i) Where am i? (ii) Where am I going? and (iii) How do I get there*? To achieve (semi)autonomous navigation, a few more processes are needed to answer these questions.

Robot navigation in the context of CARMEN is map-based navigation, where maps represent the environment in forms of metric occupancy grid maps (For detailed information on map based navigation, please refer to the Technical Report on Autonomous Robot Navigation, March 2009). Accordingly, the *map*_server process does exactly what it is named for: serving an occupancy grid map to the requesting system processes.

Similarly, the *localize* process estimates the pose of the robot in the environment. It requires a map published by the *map* process, range readings published by the *laser* process and the motion commands (odometry) published by the *robot* process. The *localize* process implements a classical Monte-Carlo (Particle filtering) scheme to estimate the pose (the position and the orientation) of the robot. As a result, this process provides the estimated pose of the robot and the associated uncertainities to the estimation.

Finally, the *navigator* process takes care of the rest of the job by planning a path to the robot from its current position to a (reachable) destination (on the map). It requests map and pose information and estimates a number of waypoints for the robot to follow. These waypoints are passed to the *robot* process which are translated to velocity vectors and further passed to the robot hardware.

The basic problem with autonomous robot navigation in real environments is the dynamic changes in the environment, which cannot be reflected to maps. Therefore the *navigator* process needs to additionally incorporate range sensor information and frequently re-plan the path based on the changing information.

3.1.1.4 Complementary Processes

The above-mentioned processes are the necessary and sufficient processes to achieve a semi-autonomously navigating robot. But CARMEN provides a few extra modules that complement the navigation problem.

The most basic of these processes is the *logger* process. What it simply does is to asynchronously record all the published information in a log file. The messages from different processes are joined, time-stamped and sequentially listed by the *logger*.

The *vasco* process (named after the famous cartographer *Vasco De Gama*) is the most distinguished of the complementary processes in the CARMEN package as it can be used to process the recorded log files to generate occupancy grid maps. It utilizes a simple scan-matching algorithm to estimate the positions of the walls and other static objects in the environment and correct the pose of the robot. That being said, some hands on experience with the *vasco* showed that it is not capable of generating maps in the environments where Nestor was tested (DTU, Bispebjerg). But luckily, an additional library called *gmapping* is made available as a plug-in to CARMEN just for this task. It is based on an "*Improved Rao-Blackwellized Particle Filter"* which is described in detail in [7]. The *gmapping* process works similar to *vasco*; it processes a log file and generates an occupancy grid map.

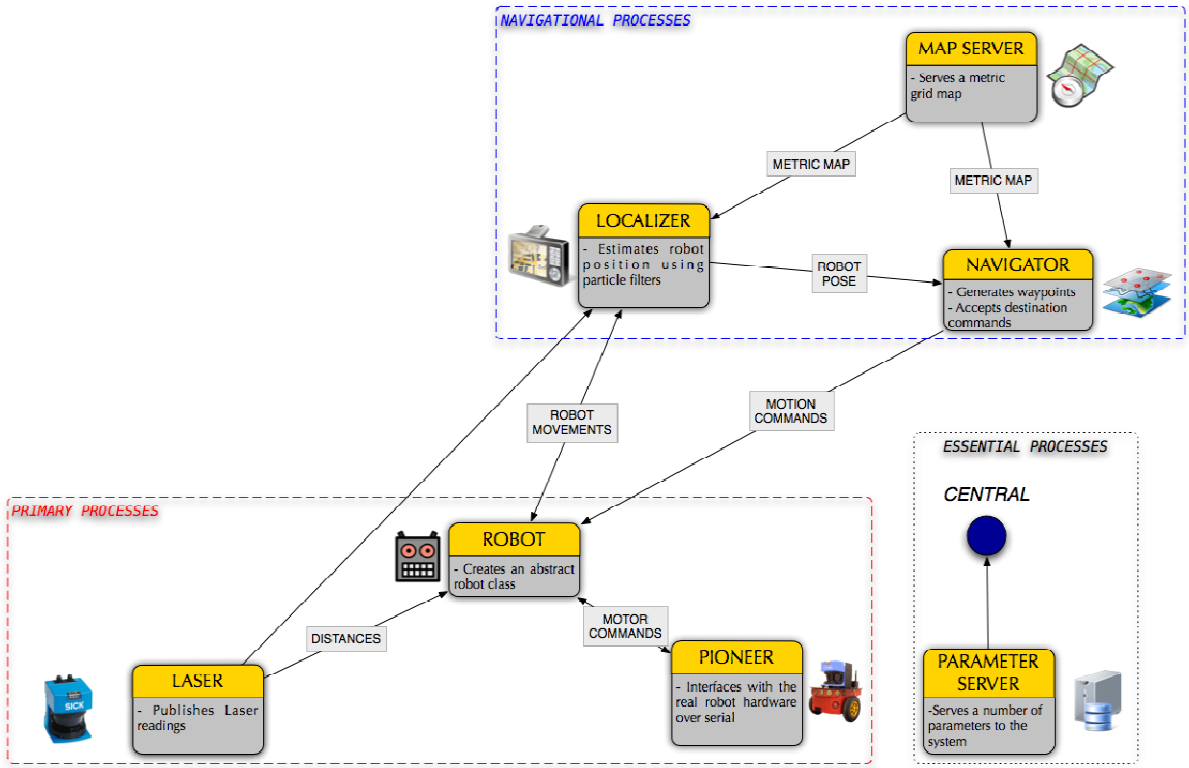

Figure 3, Navigational Processes

The *simulator* module facilitates a simulation environment to the rest of the system. Instead of using the specific robot process, i.e. the *pioneer* it can be used to simulate the robot hardware and therefore smoothly run the rest of the processes without any need for alteration.

The rest of the complementary processes can be briefly listed as:

a. *camera:* publishes images
b. *gps:* publishes gps data
c. *joystick:* interfaces with a gamepad/joystick for manual navigation
d. *pantilt*: controls an aux pan-tilt module for the camera
e. *log_playback*: playbacks recorded log files as they are recorded at the time of playback.
f. *Hmap*: provides support to hierarchical maps and transparently handle map transitions.

4 Robot Navigation with the Help of a Camera

Cameras have been extensively used in robotics for the purpose of navigation. There are numerous methods that use cameras for extracting metric information from the environment [8] or to recognize places for topological navigation [9]. To improve the existing navigation system of Nestor, a supplementary vision based topological navigation system is proposed. The underlying idea is very simple; to augment the existing metric map with a map of topological nodes so that the robot can correct its position when it encounters one of the nodes.

There have been a number of research projects that attempted to solve the topological navigation / place recognition problem using “natural” features in environments (Please refer to the technical report on Autonomous Mobile Robot Navigation for more examples). These approaches virtually rely on machine learning techniques that are based on statistical methods. They are usually needed to be trained and tested, since they basically are, classifiers.

In the Nestor project, an alternative approach is adopted, which is based on artificial markers instead of “natural” features in the environment. The major disadvantage of this approach is the need for the initial installment in the environment. On the other hand, they are much easier to distinguish and recognize, which makes the alternative approach much more reliable. Also, considering the facts that they need little or no training, they are relatively easy to install (a printed paper and a piece of tape) and they require much less computational power; it makes the artificial landmark based approach favored over natural landmark based approach.

The decision of using artificial marks for topological navigation comes with another question: What kind of artificial mark to use? This question is addressed in a separate paper: “*Visual Tags and Mobile Robots: A Review*”, where several types of artificial marks are investigated. The concluding remark of the paper can be stated as follows: It is best to use augmented reality tags as they are easy to detect using a similar setup to a mobile robot; but 2d Datamatrix barcodes might also be of interest.

Consequently, it is decided to use the markers and marker detection algorithm from the augmented reality project ARToolKitPlus [10]. Using these markers, a topological navigation system is developed on top of the existing navigation system.

The system consists of four modules: Node Map Maker, Tag-Identifier, Place Recognizer and Re-Localizer. These modules are described as follows.

4.1 Node Map Maker (Topological map generation component)

This component is used for processing metric maps and combining them together with user supplied topological node data, i.e. marker IDs. The input of the component is the generated metric map and user supplied topology information. The output of the component is a hybrid metric-node map.

4.2 Tag Identification component

This component is used for detecting the existing markers on the supplied video stream. Multiple instances of this component can be and is planned to be run for marker detection using multiple cameras. The input of the system is a video stream (or a sequence of images), and the output of the system is the ID numbers and positions of the markers on the input image.

4.3 Place Recognizer (Topological localization component)

This component is used for estimating the position of the robot using the hybrid map and the tag identification component. It simply reveals the relation between identified marker(s) and the relevant information stored in the database (topological map). The inputs of this system are the ID of the marker detected and the topological map. The output of the system is the position of the marker relative to the global map.

4.4 Re-Localizer (Integration component)

This component fuses the information given by the topological localization component and the particle-filter based metric localization process, which yield to a metric re-localization routine under the presence of an identified marker. The inputs to the component are the estimated position of the robot and the estimated position of the identified marker. The output of the component is corrected position of the robot.

Using the frameworks of CARMEN and ARToolKitPlus, these components are developed on top of the existing metric navigation system. The resulting system is graphically represented in Figure 4 (See the appendix for a larger print)

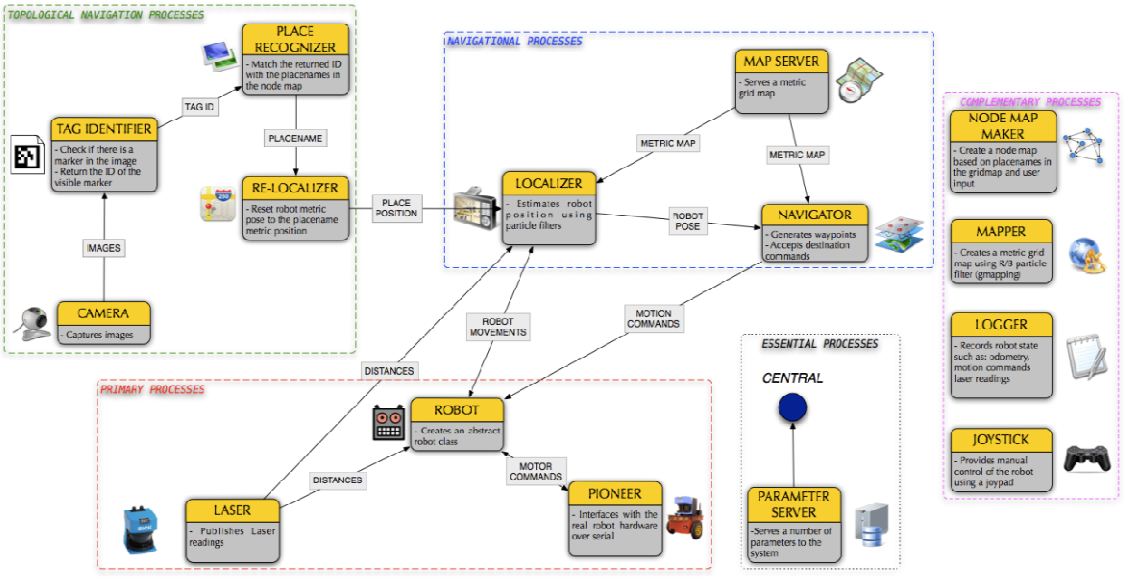

Figure 4, The developed approach

5 Experiments

The resulting system has been tested in component level throughout the development process, but to show the proof of concept two experiments were conducted.

In the first experiment, a simulated robot was used. Using the previously generated metric map of DTU, first a node map was manually generated that correlates the placenames in the metric map to the Ids of the markers that are supposed to be installed at these locations. Then, using the metric map and the generated node map, simulated robot was driven in the environment. In parallel the topological navigation part of the system is activated using a real camera and printed markers. The experiment showed that the robot successfully relocated itself to the corresponding place when a corresponding marker is detected in the video stream.

In the second experiment, a real life test was conducted, but instead of using a real robot, the computer that acts as the brain for the robot is manually moved inside a typical corridor environment. The metric map of the environment was manually drawn beforehand and its node map was generated based on the real locations of the markers installed in the environment, as shown in Figure 6.

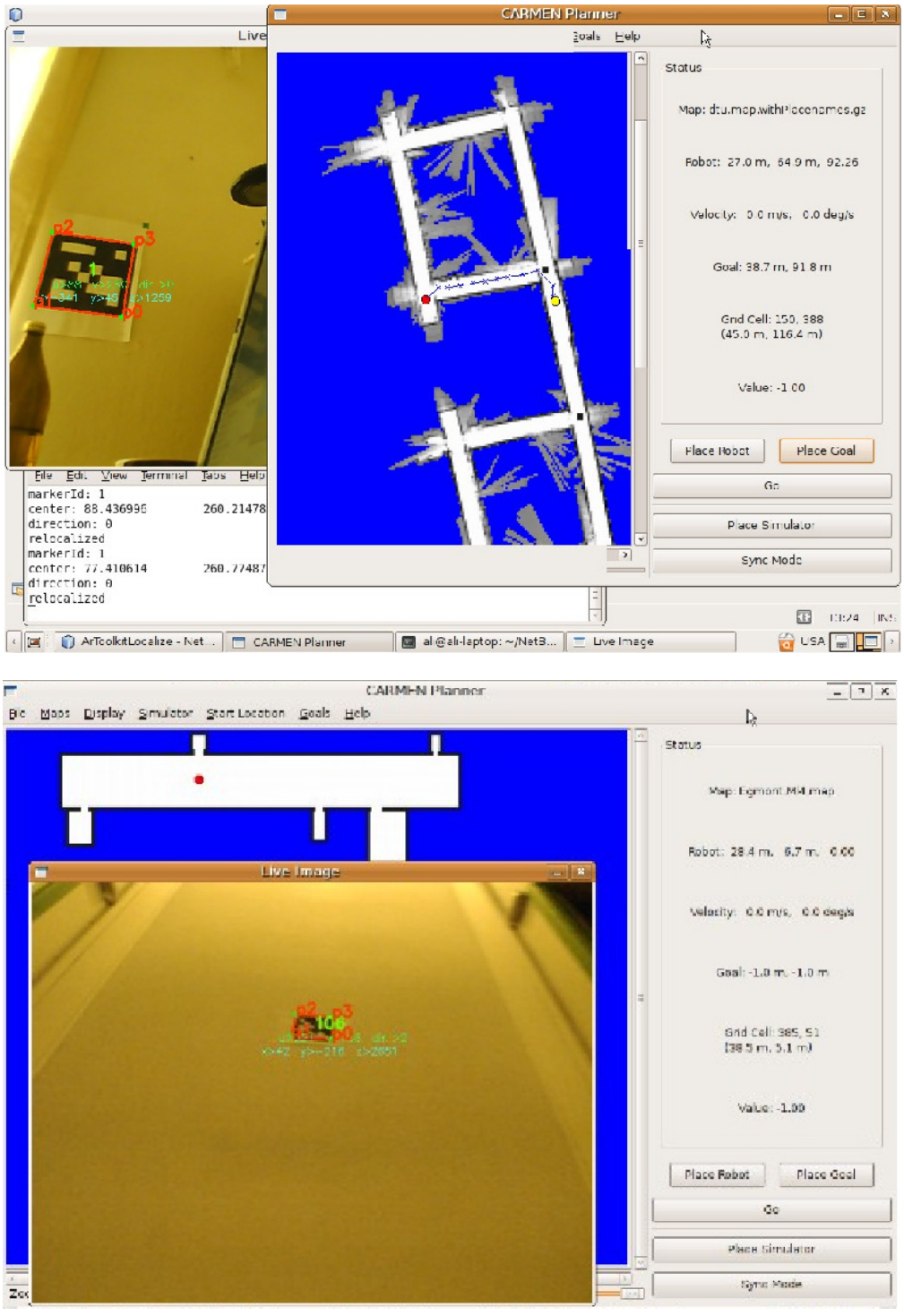

Figure 5, The moment of truth: successful relocalization in the simulation (top) and real life (bottom) experiments

Figure 6, A markers on the ceiling of a typical corridor environment

6 Conclusion

In this technical report, it is aimed to outline a new simple, yet effective approach for mobile robot localization. The new approach combines two modes of sensory information and makes use of a hybrid map that compose of a metric and a topological map. The system is built upon the existing framework of CARMEN, and it is extended with ARToolkitPlus to achieve topological navigation. The developed approach consists of four new modules that handle topological map generation, tag identification, place recognition and re-localization. The approach was subject to two initial tests that endorsed the initial claims and showed the potential use of such a system.

7 Future Work

To better show the applicability of the method, tests with real robots are needed to be conducted. These tests will also incorporate the new camera hardware, which gives significantly better quality images to process at higher frame rates. In these tests, it is also planned to use multiple cameras and assess the advantages of this approach. On the other hand, the new cameras utilize a different bus technology (ieee1394) and they require additional capturing methods, which need to be developed. The full integration of the new cameras is in the works.

One property of the augmented reality markers that has not been used is the ability to estimate the camera (robot) pose. Using this property can further improve the relocalization accuracy, as the robot can estimate its position relative to the maker. Following the tests, it will be investigated whether it is worthy to utilize this property.

As briefly mentioned above, 2d Datamatrix barcodes can be an alternative to augmented reality markers. An initial attempt showed that these markers can be decoded using live video feed from a webcam. But before implementing Datamatrix codes as alternative artificial landmarks to the system, further testing of the performance of barcode decoding should be carried out.

Datamatrix barcodes can be a powerful supplement to the topological navigation system as they can encode directly on the marker. This can even lead to mapless navigation where each marker embeds the topology information on the marker itself.

Luckily, the developed system is designed to be modular, which will let the reuse of several components in case of a transition to or addition of a new marker system.

Bibliography

[1] R. Siegwart and I.R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, MIT Press, 2004.

[2] J. Jackson, “Microsoft robotics studio: A technical introduction,” *Robotics & Automation Magazine, IEEE*, vol. 14, 2007, pp. 82-87.

[3] B. Gerkey, R. Vaughan, K. Stoy, A. Howard, G. Sukhatme, and M. Mataric, “Most valuable player: a robot device server for distributed control,” *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, 2001, pp. 1226-1231 vol.3.

[4] T.M.P.A.I.R.L. University of Malaga, “The Mobile Robot Programming Toolkit (MRPT),” *The Mobile Robot Programming Toolkit (MRPT)*.

[5] M. Montemerlo, N. Roy, and S. Thrun, “Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit,” *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings*, 2003.

[6] MobileRobots inc., “P3dx General purpose robot base,” *P3dx General purpose robot base*.

[7] G. Grisetti, C. Stachniss, and W. Burgard, “Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters,” *IEEE TRANSACTIONS ON ROBOTICS*, vol. 23, 2007, p. 34.

[8] A.J. Davison, I.D. Reid, N.D. Molton, and O. Stasse, “MonoSLAM: Real-time single camera SLAM,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, 2007, p. 1052.

[9] T. Goedemé, M. Nuttin, T. Tuytelaars, and L. Van Gool, “Omnidirectional Vision Based Topological Navigation,” *International Journal of Computer Vision*, vol. 74, 2007, pp. 219-236.

[10] D. Wagner and D. Schmalstieg, “Artoolkitplus for pose tracking on mobile devices,” *Computer Vision Winter Workshop*, 2007, pp. 6-8.

APPENDIX 1: Carmen Parameters

```
[*]
####################################################
#TEST PARAMETERS
my_parameter 1981
####################################################

# Parameters for all robots

################################
# robot parameters
robot_allow_rear_motion              off
robot_rectangular             0      # right now used only by robot_gui
robot_use_laser                      on
robot_use_sonar                      off
robot_collision_avoidance     on
robot_odometry_inverted         off
robot_interpolate_odometry      on
robot_turn_before_driving_if_heading_bigger_than    1.5708

################################
# laser parameters
laser_num_laser_devices          1
laser_use_device_locks           off
################################
# simulator parameters
################################
# camera parameters
camera_interframe_sleep         0.1
camera_dev                      /dev/video0
camera_image_width            640
camera_image_height           480
```

```
################################
# localize parameters
localize_num_particles        300
localize_laser_max_range      20.0
localize_use_rear_laser               off

## old motion model
localize_odom_a1              0.2     # a1 = error in rotation
                                      # as a function of rotation
localize_odom_a2              0.01    # a2 = error in rotation
                                      # as a function of translation
localize_odom_a3              0.2     # a3 = error in translation
                                        # as a function of translation
localize_odom_a4              0.01    # a4 = error in translation
                                        # as a function of rotation
## new (learned) motion model
localize_mean_c_d -0.0123
localize_mean_c_t -0.1065
localize_std_dev_c_d 0.1380
localize_std_dev_c_t 0.2347

localize_mean_d_d 1.0055
localize_mean_d_t 0.0025
localize_std_dev_d_d 0.1925
localize_std_dev_d_t 0.3982

localize_mean_t_d -0.0025
localize_mean_t_t 0.9638
localize_std_dev_t_d 0.0110
localize_std_dev_t_t 0.3300



################################
# navigator parameters
navigator_goal_size                   0.2
navigator_goal_theta_tolerance                0.09
```

```
##############################
# robotgraph parameters
robotgraph_rear_laser        off
##############################
# vasco parameters
# possible values: sick, samsung, urg
vasco_laser_type      sick
##############################
# linemapping parameters
# the maximum beam length, which should be used
# to extract lines
linemapping_laser_maxrange             6.0
# Split'n'Merge: max. distance of a point to a segment in the "split"-step
# (e.g.: if distance>sam_tolerance then the point set will be splited)
# with smaler values you get less line segments, but more accurate ones
linemapping_sam_tolerance              0.1
# Split'n'Merge:  max. distance of neighbouring points, so that a segment
# will be created.
# (E.g. if 'distance>sam_max_gap' then the point set will be splited)
linemapping_sam_max_gap                0.3
# Split'n'Merge: minimum length of a line segment
linemapping_sam_min_length             0.4
# Split'n'Merge: minimun number of points on a line segment
linemapping_sam_min_num                5
# use the fitting algorithm when merging lines
linemapping_sam_use_fit_split          off
# max. distance of the two end points of a line segment for merging
linemapping_merge_max_dist             0.1
# the minimum overlap between to lines before they get merged (relative)
linemapping_merge_min_relative_overlap  0.2
# the minimum overlap between to lines before they get merged (in m)
linemapping_merge_overlap_min_length   0.2
# when mergeing, distribute points over the linesegment for re-computation
linemapping_merge_uniformly_distribute_dist 0.05
```

```
##############################
# xsens (imu) parameters
xsens_dev            /dev/ttyUSB0
xsens_adapt
################################################################################
##
##  Robot-specific parameters
##
[p3]
# Parameters for Pioneer 2-DX8 Plus
base_type                               pioneer
base_model                              p2d8+
base_dev                                /dev/ttyUSB0
base_relative_wheelsize                 1.0
base_relative_wheelbase                 1.0
# base_use_hardware_integrator should be off, otherwise
# Pioneer2 (DX8Plus) ca only drive 15m in one direction
# (because of a roll-over at 0x7fff)
base_use_hardware_integrator            off
# laser parameters
laser_laser1_dev           /dev/ttyUSB1
laser_laser1_type          LMS
laser_laser1_baud          38400
laser_laser1_resolution    1.0
laser_laser1_fov           180
laser_laser1_flipped   0
laser_laser1_use_remission  none     # none / direct / normalized
laser_laser2_dev            none
laser_laser3_dev                none
# robot parameters
robot_length                        0.445
robot_width                         0.33
robot_frontlaser_use             on
robot_frontlaser_id              1
```

```
robot_frontlaser_side_offset     0.0
robot_frontlaser_angular_offset  0.0
robot_rearlaser_use              off
robot_rearlaser_id               2
robot_rearlaser_side_offset      0.0
robot_rearlaser_angular_offset   3.1415923
robot_frontlaser_offset              0.04
robot_rearlaser_offset               0.0
robot_min_approach_dist              0.15
robot_min_side_dist                  0.20
robot_acceleration                   0.80
robot_deceleration                   1.0
robot_reaction_time                  0.1
robot_max_t_vel                      0.8
robot_max_r_vel                      0.8
robot_theta_gain                     1.0
robot_theta_d_gain                   0.3
robot_displacement_gain              0.75
robot_use_sonar            off
robot_use_bumper           off



################################################################################################
#########

##

##  Expert parameters

##


[expert]


joystick_deadspot                    on
joystick_deadspot_size               0.2
localize_min_wall_prob               0.25
localize_outlier_fraction            0.90
localize_update_distance             0.20
localize_integrate_angle_deg         3.0
localize_do_scanmatching             off
```

```
localize_constrain_to_map            off
localize_occupied_prob               0.5
localize_lmap_std                    0.3
localize_global_lmap_std             0.6
localize_global_evidence_weight              0.01
localize_global_distance_threshold   2.0
localize_global_test_samples         100000
localize_use_sensor                  on
localize_tracking_beam_minlikelihood 0.45
localize_global_beam_minlikelihood   0.9
navigator_map_update_radius             3.0
navigator_map_update_obstacles          on
navigator_map_update_freespace               off
navigator_map_update_num_laser_beams    361
navigator_replan_frequency           5
navigator_smooth_path                on
navigator_dont_integrate_odometry    off
navigator_plan_to_nearest_free_point    on
navigator_waypoint_tolerance            0.3
navigator_panel_initial_map_zoom             100.0
navigator_panel_track_robot                  on
navigator_panel_draw_waypoints                       on
navigator_panel_show_particles                       off
navigator_panel_show_gaussians                       off
navigator_panel_show_true_pos                on
navigator_panel_show_tracked_objects         off
navigator_panel_show_laser                   off
navigator_panel_show_simulator_objects               off
base_motion_timeout                             1
robot_sensor_timeout                                         3.0
robot_collision_avoidance_frequency                  10.0
robot_turn_before_driving_if_heading_bigger_than_deg 90.0
robotgui_connect_distance            40.0
robotgui_gui_control                 on
robotgui_show_velocity               off
robotgui_show_vector                 on
```

```
simulator_person_leg_width           0.1
simulator_person_dist_from_robot     0.4
simulator_person_speed               0.3
simulator_dt                                 0.172
simulator_time                               0.172
simulator_sync_mode                          off
simulator_laser_probability_of_random_max    .0001
simulator_laser_probability_of_random_reading       .0001
simulator_laser_sensor_variance                     .001
simulator_sonar_probability_of_random_max    .01
simulator_sonar_probability_of_random_reading       .005
simulator_sonar_sensor_variance                     .05
simulator_use_robot                          off
simulator_frontlaser_maxrange        81      # m
simulator_rearlaser_maxrange         81      # m
camera_brightness                       -1
camera_hue                              -1
camera_saturation                       -1
camera_contrast                         -1
camera_gamma                            -1
camera_denoisestrength                  0
camera_awbmode                          custom
camera_awbred                           16384
camera_awbblue                          8192
camera_antiflicker                      off
camera_backlightcompensation            off
camera_useautosharpen                   off
camera_sharpenstrength                  49152
camera_useautoshutter                   on
camera_shutterlength                    0
camera_useagc                           off
camera_gain                             16384
camera_fps                              15
#logger parameters
```

```
##logger_ascii        off

logger_odometry              on

logger_laser          on

logger_robot_laser    on

logger_localize              on

logger_params         on

logger_gps            on

logger_simulator      on

logger_arm            on

logger_sonar           on

logger_bumper          on

logger_imu             on

logger_motioncmds      off  # includes base_velocity and motion commands given to
robot
```

APPENDIX 2: System Architecture